

FIG.1

MEM File {

```

S_O grant_o CLOCK 35 std_logic_vector (3:0) b03 BEHAV /signal-variable nom
horloge-synchronisation type taille nom-entité nom-architecture /
VAR coda0 CLOCK 35 std_logic_vector (2:0) b03 BEHAV
VAR coda1 CLOCK 35 std_logic_vector (2:0) b03 BEHAV
VAR coda2 CLOCK 35 std_logic_vector (2:0) b03 BEHAV
VAR coda3 CLOCK 35 std_logic_vector (2:0) b03 BEHAV
VAR fu1 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR fu2 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR fu3 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR fu4 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR grant CLOCK 35 std_logic_vector (3:0) b03 BEHAV
VAR ru1 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR ru2 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR ru3 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR ru4 CLOCK 35 std_logic (0:0) b03 BEHAV
VAR stato CLOCK 35 std_logic_vector (1:0) b03 BEHAV
PROCESS 1
  
```

FIG.6

MEM File {

```

S_O A_Q_OUT CLOCK 20 REG (7:0) example_4_processes
S_O B_Q_OUT CLOCK 26 REG (7:0) example_4_processes
S_O C_Q_OUT CLOCK 35 REG (7:0) example_4_processes
S_O D_Q_OUT CLOCK 44 REG (7:0) example_4_processes
PROCESS 4
BOF
  
```

FIG.7

[illegible]

FIG. 2

HDL File

VIF File

```
LIBRARY 1 ( ieee ) /type-d'entree mware-co-ligne non-av-1-entree /
USE 2 ( ieee std_logic_1164 )
ENTITY 4 b01
DECLARATION 7 ( CLOCK ) INPUT std_logic (0:0) AFFECTED_BY ( ) /type-declaration mware-co-ligne non-objet ent
DECLARATION 8 ( RESET ) INPUT std_logic (0:0) AFFECTED_BY ( )
DECLARATION 10 ( request1 ) INPUT std_logic (0:0) AFFECTED_BY ( )
DECLARATION 11 ( request2 ) INPUT std_logic (0:0) AFFECTED_BY ( )
DECLARATION 12 ( request3 ) INPUT std_logic (0:0) AFFECTED_BY ( )
DECLARATION 13 ( grant ) OUTPUT std_logic_vector (3:0) AFFECTED_BY ( )
END ENTITY 14
ARCHITECTURE 15 b01 OF b01
DECLARATION 20 ( INIT ) COM std_logic_vector (1:0) AFFECTED_BY ( )
DECLARATION 21 ( ANALIST_REQ ) COM std_logic_vector (1:0) AFFECTED_BY ( )
DECLARATION 22 ( ACTION_CONST ) COM std_logic_vector (1:0) AFFECTED_BY ( )
DECLARATION 23 ( c1 ) SIG std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 25 ( c2 ) COM std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 26 ( c3 ) COM std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 27 ( c4 ) COM std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 28 ( c5 ) COM std_logic_vector (2:0) AFFECTED_BY ( )
PROCESS 35 ( CLOCK RESET )
DECLARATION 35 ( code0 ) VAR std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 36 ( code1 ) VAR std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 37 ( code2 ) VAR std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 38 ( code3 ) VAR std_logic_vector (2:0) AFFECTED_BY ( )
DECLARATION 39 ( state ) VAR std_logic_vector (1:0) AFFECTED_BY ( )
DECLARATION 40 ( val val val val ) VAR std_logic (0:0) AFFECTED_BY ( )
DECLARATION 41 ( val val val val ) VAR std_logic (0:0) AFFECTED_BY ( )
DECLARATION 42 ( grant ) VAR std_logic_vector (3:0) AFFECTED_BY ( )
INSTRUCTION 43 IF ( RESET )
INSTRUCTION 44 AFFECT state AFFECTED_BY ( INIT )
INSTRUCTION 47 AFFECT code0 AFFECTED_BY ( )
INSTRUCTION 48 AFFECT code1 AFFECTED_BY ( )
INSTRUCTION 49 AFFECT code2 AFFECTED_BY ( )
INSTRUCTION 50 AFFECT code3 AFFECTED_BY ( )
INSTRUCTION 51 AFFECT val AFFECTED_BY ( )
INSTRUCTION 52 AFFECT val AFFECTED_BY ( )
INSTRUCTION 53 AFFECT val AFFECTED_BY ( )
INSTRUCTION 54 AFFECT val AFFECTED_BY ( )
INSTRUCTION 55 AFFECT val AFFECTED_BY ( )
INSTRUCTION 56 AFFECT val AFFECTED_BY ( )
INSTRUCTION 57 AFFECT val AFFECTED_BY ( )
INSTRUCTION 58 AFFECT val AFFECTED_BY ( )
INSTRUCTION 59 AFFECT grant AFFECTED_BY ( )
INSTRUCTION 60 ACTION state = ACTION_CONST
INSTRUCTION 61 ELSEIF ( CLOCK )
INSTRUCTION 62 CASE ( state )
INSTRUCTION 63 WHEN ANALIST_REQ =>
INSTRUCTION 64 ACTION ( c1 ) AFFECTED_BY ( code0 )
INSTRUCTION 65 ACTION ( c2 ) AFFECTED_BY ( code1 )
INSTRUCTION 66 ACTION ( c3 ) AFFECTED_BY ( code2 )
INSTRUCTION 67 IF ( val )
INSTRUCTION 68 IF ( val )
INSTRUCTION 69 AFFECT code0 AFFECTED_BY ( code0 )
INSTRUCTION 70 AFFECT code1 AFFECTED_BY ( code1 )
INSTRUCTION 71 AFFECT code2 AFFECTED_BY ( code2 )
INSTRUCTION 72 AFFECT code3 AFFECTED_BY ( code3 )
INSTRUCTION 73 ELSEIF ( val )
INSTRUCTION 74 ELSEIF ( val )
INSTRUCTION 75 IF ( val )
INSTRUCTION 76 AFFECT code0 AFFECTED_BY ( code0 )
INSTRUCTION 77 AFFECT code1 AFFECTED_BY ( code1 )
INSTRUCTION 78 AFFECT code2 AFFECTED_BY ( code2 )
INSTRUCTION 79 AFFECT code3 AFFECTED_BY ( code3 )
INSTRUCTION 80 ELSEIF ( val )
INSTRUCTION 81 ELSEIF ( val )
INSTRUCTION 82 IF ( val )
INSTRUCTION 83 AFFECT code0 AFFECTED_BY ( code0 )
INSTRUCTION 84 AFFECT code1 AFFECTED_BY ( code1 )
INSTRUCTION 85 AFFECT code2 AFFECTED_BY ( code2 )
INSTRUCTION 86 AFFECT code3 AFFECTED_BY ( code3 )
INSTRUCTION 87 ELSEIF ( val )
INSTRUCTION 88 IF ( val )
INSTRUCTION 89 AFFECT code0 AFFECTED_BY ( code0 )
INSTRUCTION 90 AFFECT code1 AFFECTED_BY ( code1 )
INSTRUCTION 91 AFFECT code2 AFFECTED_BY ( code2 )
INSTRUCTION 92 AFFECT code3 AFFECTED_BY ( code3 )
INSTRUCTION 93 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 94 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 95 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 96 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 97 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 98 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 99 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 100 AFFECT val AFFECTED_BY ( val )
INSTRUCTION 101 AFFECT state AFFECTED_BY ( ACTION_CONST )
INSTRUCTION 102 WHEN ( ACTION_CONST )
INSTRUCTION 103 IF ( val val val val )
INSTRUCTION 104 CASE ( code0 )
INSTRUCTION 105 WHEN ( val )
INSTRUCTION 106 WHEN ( val )
INSTRUCTION 107 WHEN ( val )
INSTRUCTION 108 WHEN ( val )
INSTRUCTION 109 WHEN ( val )
INSTRUCTION 110 AFFECT ( state ) AFFECTED_BY ( )
INSTRUCTION 111 WHEN ( val )
INSTRUCTION 112 AFFECT ( state ) AFFECTED_BY ( )
INSTRUCTION 113 WHEN ( val )
INSTRUCTION 114 AFFECT ( state ) AFFECTED_BY ( )
INSTRUCTION 115 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 116 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 117 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 118 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 119 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 120 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 121 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 122 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 123 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 124 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 125 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 126 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 127 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 128 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 129 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 130 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 131 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 132 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 133 WHEN ( state ) AFFECTED_BY ( )
INSTRUCTION 134 WHEN ( state ) AFFECTED_BY ( )
END CASE 135
END PROCESS 136
END ARCHITECTURE 140
```

FIG.3

```

// Example of Multiple Processes Verilog sans scan

module example_4_processes (RESET, CLOCK, ENABLE, D_IN,
                           A_Q_OUT, B_Q_OUT, C_Q_OUT, D_Q_OUT);

input RESET, CLOCK, ENABLE;
input      [7:0] D_IN;
output     [7:0] A_Q_OUT;
output     [7:0] B_Q_OUT;
output     [7:0] C_Q_OUT;
output     [7:0] D_Q_OUT;

reg        [7:0] A_Q_OUT;
reg        [7:0] B_Q_OUT;
reg        [7:0] C_Q_OUT;
reg        [7:0] D_Q_OUT;

// D flip-flop
always @(posedge CLOCK)
begin
    A_Q_OUT = D_IN;
end

// Flip-flop with asynchronous reset
always @(posedge CLOCK)
begin
    if (RESET)
        B_Q_OUT = 8'b00000000;
    else
        B_Q_OUT = D_IN;
end

// Flip-flop with asynchronous set
always @(posedge CLOCK)
begin
    if (RESET)
        C_Q_OUT = 8'b11111111;
    else
        C_Q_OUT = D_IN;
end

// Flip-flop with asynchronous reset & clock enable
always @(posedge CLOCK)
begin
    if (RESET)
        D_Q_OUT = 8'b00000000;
    else if (ENABLE)
        D_Q_OUT = D_IN;
end
endmodule
  
```

HDL File

FIG.4

BOP

VIF File

```

MODULE 5 example_4_processes { A_Q_OUT B_Q_OUT CLOCK C_Q_OUT D_IN D_Q_OUT ENABLE
RESET }
DECLARATION 7 INPUT $0:05 { CLOCK ENABLE RESET }
DECLARATION 8 INPUT $7:05 { D_IN }
DECLARATION 9 OUTPUT $7:01 { A_Q_OUT }
DECLARATION 10 OUTPUT $7:02 { B_Q_OUT }
DECLARATION 11 OUTPUT $7:03 { C_Q_OUT }
DECLARATION 12 OUTPUT $7:04 { D_Q_OUT }
DECLARATION 14 REG $7:05 { A_Q_OUT }
DECLARATION 15 REG $7:06 { B_Q_OUT }
DECLARATION 16 REG $7:07 { C_Q_OUT }
DECLARATION 17 REG $7:08 { D_Q_OUT }
PROCESS 20 { CLOCK }
SINCRO_CLK 20 { CLOCK }
INSTRUCTION 22 AFFECT { A_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 23
PROCESS 26 { CLOCK }
SINCRO_CLK 26 { CLOCK }
BEGIN_SECV 28 CLOCK
INSTRUCTION 28 IF { RESET }
INSTRUCTION 29 AFFECT { B_Q_OUT } AFFECTED_BY { }
INSTRUCTION 30 ELSE
INSTRUCTION 31 AFFECT { B_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 32
PROCESS 35 { CLOCK }
SINCRO_CLK 35 { CLOCK }
BEGIN_SECV 37 CLOCK
INSTRUCTION 37 IF { RESET }
INSTRUCTION 38 AFFECT { C_Q_OUT } AFFECTED_BY { }
INSTRUCTION 39 ELSE
INSTRUCTION 40 AFFECT { C_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 41
PROCESS 44 { CLOCK }
SINCRO_CLK 44 { CLOCK }
BEGIN_SECV 46 CLOCK
INSTRUCTION 46 IF { RESET }
INSTRUCTION 47 AFFECT { D_Q_OUT } AFFECTED_BY { }
INSTRUCTION 48 ELSE
INSTRUCTION 48 IF { ENABLE }
INSTRUCTION 49 AFFECT { D_Q_OUT } AFFECTED_BY { D_IN }
END PROCESS 50
ENDMODULE 52 example_4_processes
  
```

FIG.5

FIG.8

BEST AVAILABLE COPY

Scanned
HDL File

```
// Example of Multiple Processors Verilog

module example_4_processor (SCM_IN, SCM_IN_0002, SCM_OUT_0001, SCM_IN_0003, SCM_OUT_0002,
  SCM_IN_0004, SCM_OUT_0003, SCM_IN_0004, SCM_OUT_0004, RESET, CLOCK, M_ALE, D_IN,
  A_Q_OUT, B_Q_OUT, C_Q_OUT, D_Q_OUT);

// Declaration of some constants, some variables
// Declaration of some constants
input SCM_IN;
input SCM_IN_0002;
output SCM_OUT_0001;
output SCM_OUT_0002;
output SCM_OUT_0003;
output SCM_OUT_0004;
input SCM_IN_0003;
input SCM_IN_0004;
output SCM_OUT_0001;
output SCM_OUT_0002;
output SCM_OUT_0003;
output SCM_OUT_0004;
// Declaration of some variables
input RESET, CLOCK, M_ALE;
input [3:0] D_IN;
output [3:0] A_Q_OUT;
output [3:0] B_Q_OUT;
output [3:0] C_Q_OUT;
output [3:0] D_Q_OUT;

reg [3:0] A_Q_OUT;
reg [3:0] B_Q_OUT;
reg [3:0] C_Q_OUT;
reg [3:0] D_Q_OUT;

// Flip-Flop with synchronous clock
always @(posedge CLOCK)
begin
  if (RESET == 1'b1)
  begin
    A_Q_OUT <= 4'b0000;
    B_Q_OUT <= 4'b0000;
    C_Q_OUT <= 4'b0000;
    D_Q_OUT <= 4'b0000;
  end
  else
  begin
    A_Q_OUT <= D_IN;
    B_Q_OUT <= A_Q_OUT;
    C_Q_OUT <= B_Q_OUT;
    D_Q_OUT <= C_Q_OUT;
  end
end

// Flip-Flop with asynchronous reset
always @(posedge CLOCK)
begin
  if (RESET == 1'b1)
  begin
    A_Q_OUT <= 4'b0000;
    B_Q_OUT <= 4'b0000;
    C_Q_OUT <= 4'b0000;
    D_Q_OUT <= 4'b0000;
  end
  else
  begin
    A_Q_OUT <= D_IN;
    B_Q_OUT <= A_Q_OUT;
    C_Q_OUT <= B_Q_OUT;
    D_Q_OUT <= C_Q_OUT;
  end
end

// Flip-Flop with asynchronous reset & clock enable
always @(posedge CLOCK)
begin
  if (RESET == 1'b1)
  begin
    A_Q_OUT <= 4'b0000;
    B_Q_OUT <= 4'b0000;
    C_Q_OUT <= 4'b0000;
    D_Q_OUT <= 4'b0000;
  end
  else if (M_ALE == 1'b1)
  begin
    A_Q_OUT <= D_IN;
    B_Q_OUT <= A_Q_OUT;
    C_Q_OUT <= B_Q_OUT;
    D_Q_OUT <= C_Q_OUT;
  end
end
endmodule
```

FIG.9

BEST AVAILABLE COPY

FIG.10

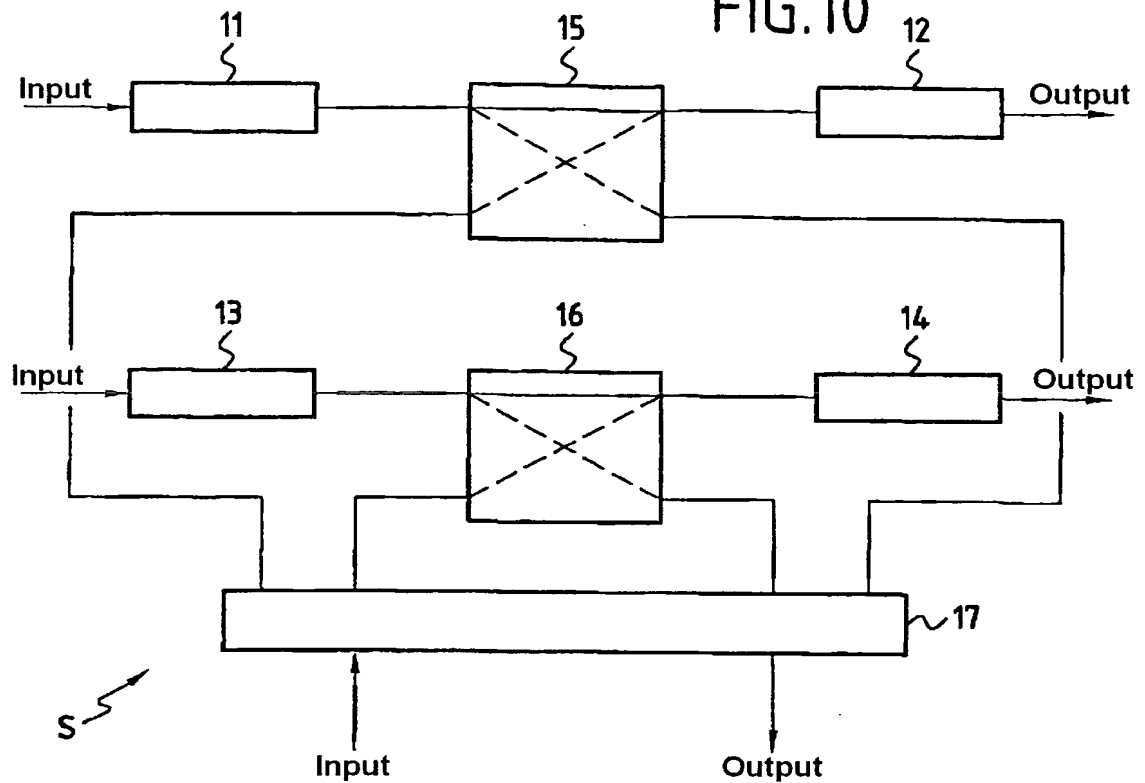


FIG.11

